Supporting Secure Ad-hoc User Collaboration in Grid Environments

Markus Lorch, Dennis Kafura
Department of Computer Science
Virginia Tech[1]
Contact e-mail: mlorch@vt.edu

Abstract

We envision that many usage scenarios involving computational grids will be based on small, dynamic working groups for which the ability to establish transient collaboration with little or no intervention from resource administrators is a key requirement. Current grid security mechanisms support individual users who are members of well-defined virtual organizations. Recent research seeks to provide manageable grid security services for self-regulating, stable communities. Our prior work with component-based systems for grid computation demonstrated a need to support spontaneous, limited, short-lived collaborations. Such collaborations most often rely on shared or delegated fine grained access privileges to data and executable files as well as to grid compute resources. The mechanisms we are developing focus on the management and the enforcement of fine grained access rights. Our solution employs standard attribute certificates to bind rights to users (or their surrogates) and enables the high level management of such fine grained privileges which may be freely delegated, traded, and combined. Enforcement is provided by POSIX operating systems extensions that extend standard file permissions and regulate resource usage through access control lists. These extensions are available for common platforms and fully support legacy services.

In combination, our privilege management and enforcement mechanisms are compatible with and enable the usage of fine-grained rights, leverage other work in the grid computing and security communities, reduce administrative costs to resource providers, enable ad-hoc collaboration through incremental trust relationships and can be used to provide improved security service to long-lived communities.

1. Introduction

This paper describes two mechanisms to support secure collaboration among ad-hoc, transient groups of grid users with minimal overhead imposed on resource administrators. Currently deployed grid security services support stable, well defined collaboratories [PEA01] and virtual organizations [FOS01] through single sign-on, rights delegation, the integration of local security solutions, and retention of final authority over the use of resources by the resource owners. Often only coarse grained access decisions are made by these grid authorization services (e.g. the mapping of authenticated global users to local user accounts as done by the Grid Security Infrastructure GSI [FOS98] and in UNICORE [ROM00]). The resulting requirement for existing local user accounts on the grid resources for every grid entity poses a scalability problem and hinders ad-hoc collaboration.

Current research aims at supporting long-lived communities in which the allocation of access rights is broken into two levels. The allocation of rights to a community as a whole by resource providers (resource administration) and the allocation of rights to individual members of the community by community administrators (community administration). This separation reduces the administrative costs for resource owners and enables community-specific, fine grained security policies. Our work seeks to further extend this line of research by supporting collaborations that are unanticipated (achieved by dynamic discovery or synchronous interaction), short-lived (perhaps only days or even a single session), and involving a small number (perhaps only two) of grid users. Our goal is it to provide a mechanism that allows for the management of fine grained rights through the grid middleware (privilege management) and a mechanism to enforce access with fine grained rights at the resource. By fine grained rights we mean an arbitrary subset of an entities rights, for example read access to a single file on a grid resource.

[1] This research is funded by the Virginia Commonwealth Information Security Center (CISC) http://www.cisc.jmu.edu/.

The focus on ad-hoc groups arose from our earlier work with Symphony [LOR02], a component-based system for collaboration and sharing. Symphony enables users to integrate components representing grid computations or data made available by others. The integrated components can be accumulated through resource discovery (asynchronous sharing) or through real-time interaction (synchronous sharing).

An ad-hoc collaboration is illustrated by the following scenario: A university researcher, Bob, has developed an experimental network protocol emulator that runs on a special purpose compute cluster at the university. John, a corporate researcher, has heard from Bob's simulator and would like to use it to simulate a new, proprietary protocol that he developed. He contacts Bob and asks for permission to use the simulator. Bob agrees, creates a privilege credential bound to Johns identity which grants short-lived access and execute permissions. Bob sends this credential (embedded in a customized Symphony component abstracting the simulator) to John e.g. by e-mail. John can now contact Bob's compute cluster, authenticate with his own identity credential and provide the privilege credential to the grid resource (the cluster). The resource will validate that the privilege credential came from an authoritative user (Bob) and grant access for the validity period of the privilege credential.

In a traditional grid security environment such a scenario would require the following steps:

1. An administrator at Bob's university creates a user and a group account for this collaboration and enters the new account into a grid access control list (grid-mapfile).
2. Both the new user (Bob) and the simulator executable would have to become members of this group. Expressiveness limitations of the operating system (generally a UNIX derivative) would prevent the simulator executable from being a member of more than one group.
3. When the collaboration ended (possibly after just a single access by Bob), the administrator would have to delete Bob's account and the group account to prevent unauthorized use in the future.

In practice, shared accounts are often used to reduce the administrative overhead. Users as well as files are members of a universal system group account. This practice reduces overall system security by preventing least-privilege access to resources and hinders ad-hoc collaboration due to the high level of trust required before such a large set of permissions can be granted and the associated administrative costs justified.

We envision that on computational grids many collaboration scenarios will be based on small, ad-hoc and dynamic working groups for which the ability to establish transient groups with little or no intervention from site administrators is a key requirement. While our work is focused on ad-hoc collaboration, the mechanisms we develop can also be used to provide improved security services to virtual organizations and other longer-lived communities as well.

The next section presents in more detail the management and enforcement issues and gives an overview on related projects. Section 3 explains our approach to provide privilege management and enforcement in grid environments. Section 4 is a description and evaluation of a prototype implementation of the proposed mechanism. Section 5 identifies research issues for future work. Section 6 concludes with a summary.

2. Issues and Related Work

Our work and two related systems will be described in relation to the closely related issues of management and enforcement of fine grained access privileges.

Management of fine grained privileges comprises functionality for the secure association (binding) and administration of privileges with grid entities. This includes the specification, assignment, delegation, composition, and revocation of fine grained security privileges in a platform independent manner. The scope of such privileges can range from single objects to multiple grid resources and entities of separate administrative domains.

The enforcement of fine grained access rights is defined as the limitation of operations performed on resources by a user to those permitted by an authoritative entity. Enforcement mechanisms need to be able to cope with expressiveness limitations of prevalent operating system security mechanisms. Portability, support for legacy applications and acceptable overhead are other key factors.

The most widely deployed grid security systems do not provide privilege management mechanisms but rather require privileges to be administrated using legacy operating system commands and tools. These systems rely on standard OS enforcement mechanisms which are not meant to support users from separate administrative domains nor do these mechanisms provide adequate support for temporary users with only a very constrained set

of fine grained rights. For example it is not possible in traditional UNIX based systems to have two groups of users share access to a specific file unless access is permitted for all users of the system. It is also not possible to give file access permissions to an additional single user without that user being a member of a user group that has access. These expressiveness limitations [PEA01] prevent many collaborative usage scenarios. More flexible and scalable means for privilege management and more powerful, fine grained enforcement mechanisms with support for legacy codes are needed to provide support for various forms of team collaborations on grid systems.

Two projects that have addressed these issues to varying extents and with different aims are the Community Authorization Service CAS [PEA01] and Akenti [THO99].

Privilege management is handled differently in CAS and Akenti. CAS incorporates authorization information into restricted GSI proxy certificates (PCs). The PCs specify an entity's subset of rights from the set of rights assigned to the CAS community. Such a PC is generated at the request of an authenticated user by a CAS server and subsequently used by the user to provide for authentication and authorization when accessing grid resources (a user never authenticates directly to a resource but rather uses the restricted community identity known to the resource). In CAS the applicable access policy for a specific request is defined by the rights granted to the community account by the resource owner less the restrictions for this access specified by the CAS administrator and embedded in the PC by the CAS server. The main goal and advantage of CAS is scalability: each resource owner assigns only coarse grained rights to a single community group account, CAS administrators manage the assignment of fine-grained privileges to community members via the CAS server.

Akenti binds user attributes (e.g. a user's role and group membership) and privileges through attribute certificates (ACs) and thus separates authentication from authorization. Akenti uses X.509 identity certificates for authentication and a set of proprietary ASCII ACs that hold policy information for privilege management. Long-lived ACs convey group and role membership status. Short-lived ACs are used to cache and share authorization decisions among processes. So called "use-case" ACs are generated by resource owners to define usage policies for their resources. In Akenti the applicable access policy for a specific request results from the intersection of use-policies defined by the resource owners and the group and role assignments made in attribute certificates by user managers. Akenti's main goal is the ability for often multiple resource owners (in Akenti referred to as stakeholders) and administrative parties to define fine-grained and flexible usage policies in a widely distributed system.

Both CAS and Akenti use policy languages that allow arbitrarily fine grained specification of privileges. Akenti relies on its own policy language [THO01] to specify privileges, use-cases and attributes. While the current CAS implementation uses a simple proprietary policy language, the CAS architecture is independent of a specific language. A variety of policy languages exist which can be used to specify privileges, user attributes and the like ranging from basic privilege lists to sophisticated languages like the extensible access control markup language (XACML) [XACML] and the digital rights language for trusted content and services (XrML) [XRML].

For enforcement, both CAS and Akenti rely on the software providing the requested service to self-regulate system access. The CAS team developed an implementation of the Generic Authorization and Access-Control API (GAA API) [RYU99] to provide a defined interface for the service to query authorization data from CAS independent of the used policy language. Akenti provides proprietary means by which an Akenti service can query for authorization information and then make access decision. The resource server (e.g. an ftp server or a scientific application) that accepts and serves a request enforces the applicable access policy.

The solutions presented by CAS and Akenti are aimed at long-lived collaborative environments that change in size and composition rather slowly and use code specifically developed for, or ported to interact with, their security mechanisms. CAS reduces the administrative overhead that basic GSI mechanisms impose on large collaborative grid communities. Akenti allows multiple resource owners to specify usage policies in a flexible manner and provides support for role based access control. The use of policy languages in both projects provides for the necessary granularity in the specification of privileges.

However, both CAS and Akenti have two shortcomings. First, neither system provides the flexibility in their privilege management mechanisms that is required by ad-hoc, short lived collaborations. Second, both systems use an incorrect layering of the enforcement mechanisms: to solve the expressiveness limitations they do not rely on operating system enforcement but rather require fine-grained enforcement of access rules at the grid-middleware and application layer. This layering violates the fundamental design principles of "separation of concerns" and "least privilege access". As a result the usage of legacy applications and standard system services in these grid security environments is not supported. Moreover, there is lower overall security due to the need for fully trusted application code which performs enforcement.

To further promote the evolution of grid security mechanisms to support short-lived, ad-hoc collaborations additional mechanisms for privilege management and enforcement are needed. It is these mechanisms at which our work is aimed. Our efforts are coordinated with other researchers from the Global Grid Forum as well as the IETF PKIX group to prevent redundant work and provide for the interoperability of our contribution.

3. Support for ad-hoc collaboration

We focus on two mechanisms: a privilege management scheme based on attribute certificates that allows for flexible assignment, delegation, composition and revocation of fine-grained rights directly by users, and a portable, low-overhead enforcement scheme for fine-grained security policies that provides support for legacy applications. Both mechanisms can be used independently of each other and can interoperate with existing security solutions. We have implemented a first prototype by extending our Symphony Framework which is discussed in section 4. Currently we are working on interfacing our mechanisms with the Grid Security Infrastructure (GSI) and Globus services [FOS99]. Our evaluation of the security mechanisms implemented by other projects, such as UNICORE and Legion [GRI99], suggests that our schemes can also be incorporated into those projects.

Our privilege management scheme employs privilege credentials in the form of standardized X509v2 attribute certificates [RFC3281] to convey privileges separately from identity credentials used for authentication. The separation of authentication from authorization, as shown by the Akenti project, allows for very flexible delegation and the combination of privileges from different sources and with variable lifetimes. By delegating and trading such credentials, users can form and dissolve groups directly through user-to-user interaction and peer trust relationships without the need for administrator intervention.

Our novel enforcement mechanism enacts fine-grained access policies through the use of commonly available, operating system extensions providing access control lists. This mechanism enables the secure execution of legacy codes, imposes significantly less overhead than mechanisms which make access decisions on a call-by-call basis (e.g. sandboxing), and provides a simple and thus reliable means to protect system resources. By linking the grid access control services to the security services provided by the underlying resource operating systems our mechanism satisfies a security requirement stated in the recent work towards an Open Grid Services Architecture [FOS02].

3.1 Privilege Management

In our mechanism privileges are specified in a platform independent policy language and are securely assigned or delegated to entities by embedding them in attribute certificates (ACs). Figure 1 shows a sample attribute certificate. We use ACs to bind privileges to a distinguished name (DN), the privilege holder. The grantor (issuer) of the privilege will sign the AC. Given the identity certificate of the issuer a resource can check if the issuer is authoritative for the embedded privilege (i.e. the owner of the corresponding system object) and determine whether the delegation is valid. The use of X.509v2 attribute certificates as defined in [RFC3281] allows us to create and process such certificates with standard tools.

The separation of privilege credentials from identity credentials avoids the limitations of impersonation. Impersonation, as used by the GSI through proxy certificates, allows an entity to delegate all or a subset of its own privileges to an agent that then acts on the delegating entity's behalf. Impersonation schemes implement a subtractive security solution – the delegated rights are obtained by reduction from the set of all rights possessed by the grantor. This solution bears the danger of violating the least privilege principle [SAL75]. As with all subtractive security schemes, it is problematic to ensure that only the minimum subset of required privileges is delegated (the entity's total rights need to be known and accurately restricted). Furthermore it may be difficult to audit who is actually using a specific resource as delegated identities are being used for authentication (E.g. a collaborator authenticates with the identity of the privilege grantor, the collaborator's own identity is not supplied to the remote resource).

With separate credentials for privileges and identities, privileges from arbitrary sources can be securely combined to create a list of minimal privileges supplied with a specific access, which is consistent with the least privilege principle. This is a much needed feature which is not supported by impersonation schemes. An entity in a user-to-user collaboration scenario needs the ability to combine a subset of its own rights with rights received from collaborating entities. Collaboration may start with a low level of trust and minimal shared privileges. As the entities continue to work together the trust level increases and additional privileges may be granted. The

ability to create flexible and dynamic combinations of privileges models more accurately the actual relations between the collaborating entities.

```
[Acinfo:
    Version: 1
    Holder:
        BaseCertificateID: null
       EntityName: [CN=Dennis Kafura, OU=Virginia Tech User, OU=Class 1, O=VT, C=us]
        ObjectDigestInfo: null
    Issuer:
        IssuerName: [CN=VT Campus Grid Admin, OU=Virginia Tech User, O=VT, C=us]
        BaseCertificateID: null
        ObjectDigestInfo: null
    Signature: SHA1withRSA, OID = 1.2.840.113549.1.1.5
    SerialNumber: SerialNumber: [    05]
    AttrCertValidityPeriod:
        NotBeforeTime: Sat May 11 09:27:16 EDT 2002
        NotAfterTime: Sat May 18 09:27:16 EDT 2002
    Attributes: 1
    [1]: Type: ObjectId: 1.3.6.1.4.1.6760.8.1.1  Values: 1
        Values[0]: AccessPrivilege://zuni.cs.vt.edu
    IssuerUniqueID: 10110101
    Extensions: null
]
SignatureAlgorithm: SHA1withRSA
SignatureValue: <omitted>
```

Figure 1 - An Attribute Certificate that conveys temporary system access privileges

Several mechanisms can be used for revoking attribute certificates. Certificate revocation lists (CRLs) are a common way to distribute information on certificate validity. The Online Certificate Status Protocol OCSP [MYE01] provides an alternative mechanism for revocation. In order to apply these means to a large number of possible issuers (as every user is authoritative for his own resource objects and thus can issue access privileges) a centralized certificate status server is needed which acts as a repository for CRLs and can answer OCSP queries. To revoke an attribute certificate, a user creates and submits a new CRL to the certificate status server. Alternatively, if the revocation is for an attribute certificate that only applies to a small number of resources, the authoritative user can push the revocation information by sending unsolicited OCSP responses directly to the effected resources.

3.2 Enforcement

Enforcement strategies can be characterized in one of three ways depending on how the identity credentials and the privilege credentials (if any) are combined to determine the effective privileges applicable to authorize a request. The three strategies are:

- **Credential Mapping:** Basic or restricted identity credentials are mapped to local user accounts. This grants all permissions associated with the local user account less the optional restrictions to the request if restricted credentials are being used. The GSI and CAS use this scheme.

- **Mixed Mode Authorization:** Credential mapping is used to define an initial set of privileges. Additional privileges based on presented privilege credentials are applied to the specific local user account before the request is served. After a resource access is completed the resource server revokes the additional privileges. Alternatively the privileges could remain assigned to the local user id for the lifetime of the presented privilege credentials.

- **Full Credential Combination:** A user requests services from a resource where the user has no permanent local account. The resource accumulates all the presented privileges and applies them to a generic, initially very restricted, user account; the request can then be served. After completion the privileges are removed and the account returned into its original, restricted state. It would also appear feasible to keep this temporary account assigned for subsequent requests based on the credential lifetimes. This type of dynamic user assignment and access authorization can be grouped with a demand-driven user allocation scheme for grid environments as suggested in [HAC01].

The enforcement of complex security policies which cannot be translated into credential mapping is a challenging task. CAS and Akenti deal with this issue by requiring a grid service to implement self-policing mechanisms that perform access control based on provided policy information. The underlying operating system no longer performs fine grained authorization of system access as these services run with a large set of OS privileges. Legacy services cannot be accommodated as they are unable to make the necessary authorization decisions (they rely on the OS restricting their access) and would thus pose a significant security risk.

We have investigated the use of file system access control lists (ACLs) on a number of UNIX operating systems, specifically Linux, Solaris, and IRIX. On these systems ACLs are implemented following the POSIX.1E [POS88] recommendation. The application of shared access privileges in many operating systems traditionally involves the creation of user group accounts and the changing of group membership for users as well as resource objects (i.e. files). File system ACLs extend the standard file permissions to allow finer control on a user-by-user basis. Figure 2 shows a typical ACL listing specifying additional access permissions for users abazaz, kafura and akarnik for a file owned by user mlorch.

```
# file: data/simparam
# owner: mlorch
# group: users
user::rw-
user:abazaz:rw-
user:kafura:rw-
user:akarnik:rw-
group::r--
mask::rw-
other::r—
```

Figure 2 – A file system ACL

ACLs can be modified dynamically by the resource gatekeeper during the authorization of an incoming request to reflect permissions stated in privilege credentials. The requested service can then be provided by a legacy program. After the request has been served the additional privileges can be revoked. This scheme provides the means necessary to assign transient access privileges to entities and support scenarios where several users have access to a single file without being members of the same user group. Dynamic, collaborative access to resource objects with a minimal set of rights assigned on demand is thus possible.

Additional privileges or restrictions that limit the use of system wide resources can also be applied and enforced by the operating system through portable and commonly available operating system tools. Quota mechanisms can be handled in this way. For example, storage space can be allocated based on conveyed ACs.

The combined use of credential mapping mechanisms together with authorization mechanisms based on cumulative privileges conveyed through ACs from possibly different issuers allows for very flexible and efficient authorization services that can operate within existing infrastructures. Two strengths of this approach are: (1) the low overhead due to enforcement within the operating system kernel, and (2) the ability to execute legacy code without modifications or wrappers. Unfortunately, the POSIX.1E recommendations have never been formally completed and standardized. Interfaces and implementations differ slightly. A portable and unified API is required to set and modify entries through grid gatekeepers. We have implemented such a library for the Linux and IRIX operating systems as part of our prototype and will soon be able to support Solaris platforms.

Two other mechanisms to enforce fine grained system access with support for legacy applications are “sandboxing” and the redirection of library calls. While it would be possible to replace our ACL based enforcement mechanism with either one of these two approaches, we prefer the ACL approach for the reasons given below.

Sandboxing of applications is another way for resources to enforce fine-grained security policies for legacy codes. Tools like Janus [GOL96] or the Virtual Execution Environment VXE [VXE02] provide a constrained environment that monitors and evaluates all system access (e.g. through system call interception). Arbitrarily fine grained access policies can be enforced using such mechanisms. However, this flexibility comes at the price of considerable overhead which is often unacceptable for high-performance applications. In addition, it may be a significant effort to configure minimal access permissions for complex codes. Finally, sandboxing environments are often closely tied to vendor specific operating system mechanisms and not portable.

Redirection of library calls through dynamically injected shadow libraries is another mechanism that can be used to enforce security policies without application support. Bypass [THA01] uses such an approach as part of the Condor [BAS97] project. This approach is more promising than sandboxing as the shadow libraries are likely to be more portable. However, it requires a mechanism to ensure that the application is not using any system calls that are not “shadowed” and thus not controlled.

4. Implementation

Symphony [LOR02] is a component-based framework for creating, sharing, composing and executing grid applications and elements thereof. The Symphony framework is implemented in Java and uses the Java CoG Kit [LAS01] to interface with Globus [FOS99] and the GSI. Symphony also features a proprietary backend resource server which uses Java RMI as its communication mechanism. Figure 3 shows a sample meta program in Symphony. An input file is read by a simulator component which generates a simulation result file which in turn is eventually processed by a visualization component. We extended the Symphony framework with our mechanisms for privilege management and enforcement. Extensions include new tools for the creation and management of attribute certificates, the ability to attach ACs to Symphony components and modifications of the Symphony server to parse and apply those ACs.

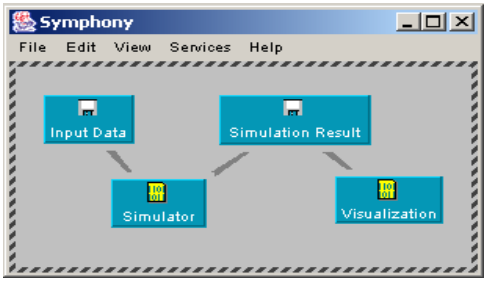

Figure 3 - A Symphony meta program

A graphical user tool to create attribute certificates that convey specific file and general system access permissions to another entity (Fig. 4), has been implemented. The "Issuer" and "Holder" entities are defined through their X.500 distinguished names (DN). The issuer can use existing GSI proxy credentials for the signing. In this case the issuer field holds the proxy credential DN and the proxy certificate path identifying the issuing end entity is stored with the AC. The holder DN can either be acquired by browsing, searching an LDAP server, or through direct interaction between the users (e.g. through e-mail). A validity period states the time frame during which the AC will be accepted by grid resources for authorization. Of course, the identity credentials used to sign the AC also determine the AC's lifetime. If a short-lived proxy credential is used for signing the AC lifetime is significantly constrained. Conveyed privileges are specified as an embedded ASCII text string. A URL like naming scheme is used, which denotes the type of privilege (FilePrivilege or AccessPrivilege), the host to which this privilege applies and, for file privileges the path to the file and a comma separated list of possible access rights (read, write, and execute). In future implementations this scheme will be extended to include a more powerful policy language.

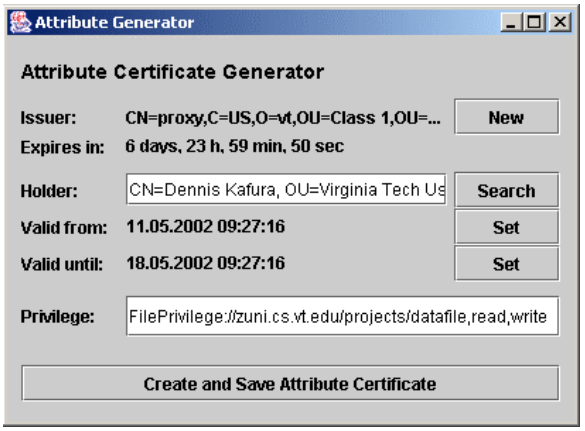

Figure 4 – The Symphony AC Generator

Created attribute certificates are saved in a PEM formatted file [RFC1421] together with the certificate path of the issuer. An encapsulation boundary of "----- BEGIN ATTRIBUTE CERTIFICATE -----" is being used to distinguish ACs from identity certificates, which are denoted by the boundary "----- BEGIN CERTIFICATE -----". PEM certificate files are easily shared among users (e.g. via e-mail). Figure 1 shows a sample attribute certificate in clear text.

Figure 5 shows the GUI used to attach attribute certificates, together with the holder's identity credentials (e.g. a GSI proxy), to a Symphony component that represents, for example, an executable program on a remote resource. When such a Symphony component is ready to be executed it presents all these credentials to the remote resource for authentication and authorization.

On the remote resource side our prototype employs a Symphony proprietary back-end server based on Java RMI which provides gatekeeper functionality. This server traditionally used PKI identity certificates to authenticate and authorize users, a new extension

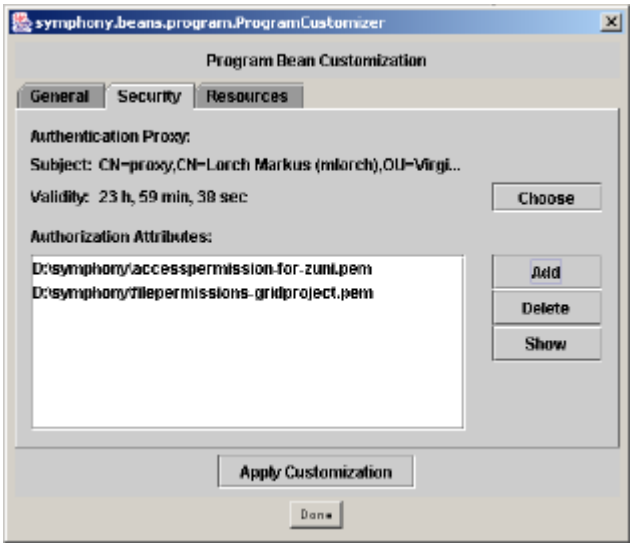

Figure 5 - Associating credentials with a Symphony component

allows it to also parse ACs and apply the embedded privileges before commencing the actual remote process.

Figure 6 shows the simplified logical program flow for authorization (authentication is also represented as the first step for completeness). The authorization procedure can be split into three logical blocks:

1. parsing of the supplied privilege credentials (ACs) and validation of authoritativeness of the issuer
2. enactment of the applicable changes to the file system ACLs
3. restoring of the original privilege state (after the request has been served)

By changing the effective user id of the resource gatekeeper when enacting privilege changes to the user id of the validated privilege issuer the system avoids security threats associated with the changing of system privileges by a process running with super user privileges. Privileges are applied on the behalf of the issuing entity and thus should be performed with only the set of rights that the issuing entity holds.

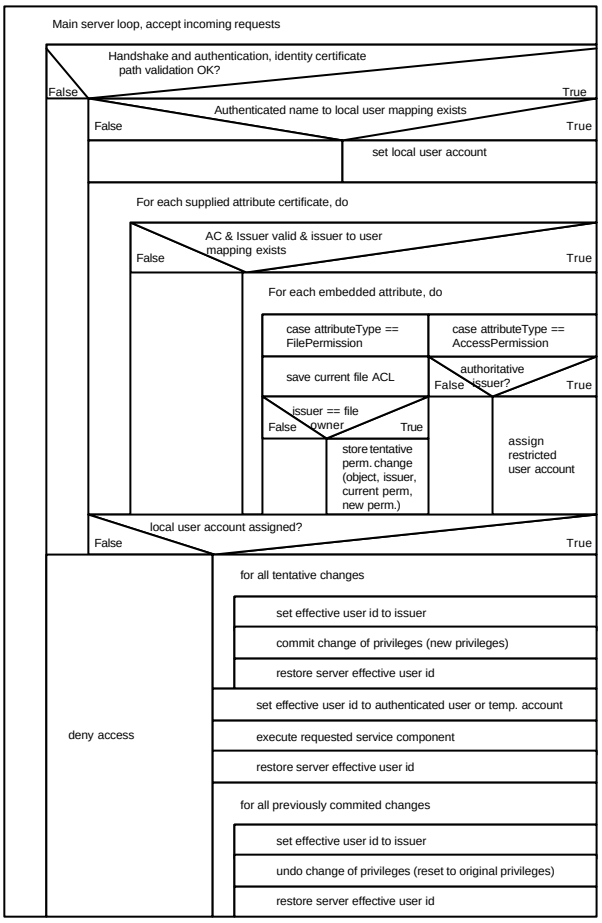

Figure 6 – Authorization using attribute certificates

The scheme shown in Figure 6 supports authorization based on credential mapping, mixed mode, and full credential combination. If an authenticated user has a local account the mapping will be performed and optional privileges credentials are evaluated and applied. If a user has no local account at the resource from which a service is requested an access privilege credential needs to be presented with the request. The resource will check if the issuer of the access privilege is authoritative to permit additional users to access the system. This capability can be given to any existing system user by the system administrator through an entry in a server configuration file. If the access privilege is valid the server maps the requesting user to a temporary, very restricted account. The "nobody" account is an example. Additional privileges need also be present in the request such that the actual service requested from the resource can be accessed (e.g. execute permission on a specific program). In future implementations we will include dynamic user management that will map grid users without a resource user account to a user account from a pool of available accounts (as done in the Legion system [GRI99]).

A native C library is used to access POSIX system calls to modify file system ACLs (see Fig. 2) and to change the effective user ID from within the Symphony server, which is implemented in Java. In order to have a portable implementation, only those system calls to modify ACL entries that are available on multiple platforms are being used by the library. The current implementation supports ACLs on Linux 2.4 and IRIX 6.5; support for Solaris will be added shortly. Solaris implements ACL system calls based on an older draft standard of POSIX 1E which requires a redundant implementation of the functions in our library to cope with a different syntax for the system calls and ACL data structures.

The native library includes functions to query and set the existing ACL associated with files and directories. The library also provides access to functions that convert ACLs between the system representation and a text representation. Figure 2 shows the long text representation. A shorter text representation is also defined and used internally by our implementation. POSIX does not define mechanisms to add or remove entries to an ACL but rather only provides means to load and write complete access control lists. It was thus necessary to implement mechanisms that parse ACLs and insert, delete or replace specific entries. An ACL consist of two types of entries: the basic entries (owner, group, other) are defined by traditional UNIX file permissions and cannot be removed; additional entries can be added for users and groups. When an ACL has additional user or group entries a "mask" entry is required. The mask entry defines the maximum effective privileges an additional entry in the ACL can grant. The Symphony server checks for the existence of such a mask entry and will not modify it if present. If no mask entry is set a new mask entry with the same privileges as the additional entry that should be added based on the processed privilege credentials is created. This feature allows the file owner to quickly revoke all privilege credentials by simply modifying or adding a mask entry with no rights

In our initial scenario example in which Bob, a university researcher, wanted to grant access to his simulator program to John, the following simple steps need to be performed when using our prototype:

1. Bob creates an attribute certificate for John, permitting him to access to the cluster while not having a local user account, plus a second AC which grants John execute permission for the simulator program. (Both privileges could also be encapsulated in one AC)
2. Bob sends the ACs to John by e-mail.
3. John accesses the cluster resource and requests the simulator service using his identity plus the ACs issued to him by Bob.

All Bob needs for our mechanism to allow this delegation of privileges is file ownership of the simulator executable and the capability to grant system access to users without a user account on the cluster resource.

We envision that system administrators will grant the capability to delegate system access permissions for grid users without a local user account on the specific grid resource to those users who normally could request the creation of a new user account on the resource. This would ordinarily include the resource owners, group or project leaders, and similar principals.

5. Discussion, research issues and future work

The first experiences with our prototype suggest that our approach is a viable solution for small, temporary and relatively unstructured collaborations where individuals need to grant each other access to their resources. An immediate goal is to incorporate our mechanisms into GSI and the Globus Toolkit. This requires the code component dealing with credential mapping in the Globus Resource Allocation Manager (a.k.a. gatekeeper) to be augmented by a module that implements the authorization procedure shown in Figure 6. We currently evaluate different ways in which privilege credentials will be presented to the remote resource in GSI. Options include the transmission during the handshake phase (as done by our prototype), the extension of the Globus Resource Allocation Management (GRAM) protocol to negotiate and transmit privilege credentials after a secure connection has been established, and well as the use of the Metacomputing Directory Service (MDS) to serve as credential repositories and be queried by the resource during authorization.

We are aware that our approach of frequently setting and revoking file privileges may cause problems with concurrent access as privileges may be revoked by one process (which has completed) while still in use by another. For example the following scenario could lead to such problems: A service is requested by a user that requires the setting of an additional file access privilege. A second service, which normally requires setting the very same access privilege is requested while the first requested service still runs. The second service will not modify the ACL as all required privileges are already present and start executing. If the first service now completes its work and resets the ACL to its original, restricted state before the second service completes, then the second service could fail as necessary permissions have been revoked unexpectedly. We are investigating several possible solutions to this issue including the introduction of a reference count on dynamically modified privileges. An approach which would mitigate concurrency problems is not to immediately revoke additional file privileges but rather to have them applied at the resource for their full lifetime. To implement this system we envision an ACL monitor daemon which watches over all changes to object privileges and periodically removes expired privileges from file ACLs. One drawback of this approach is the introduction of yet another privileged daemon process at the grid resources.

In order to create complex privilege statements that can apply to multiple resources a more powerful policy language is needed. The Akenti policy language with its hierarchical resource naming scheme as well as the Extended Access Control Markup Language (XACML) appear to be promising choices. The CAS project is also experimenting with different policy languages. We will closely monitor their results and investigate the feasibility to employ and implementation of the GAA-API and thus have a policy language independent security service as done by CAS.

Experiences with the Akenti system have shown that policy languages are too complex to be used directly by the end user to specify fine grained permissions. A user interface needs to be developed that enables an arbitrary user to specify attribute certificates that convey the minimum amount of privileges needed for a specific service. The user needs to be guided through the process of AC generation by the software tool. Furthermore, a protocol for entities to negotiate the necessary capabilities for a certain request would enable automatic credential selection. A user agent can then select the minimum necessary privileges to supply with a request from the set of privilege credentials available. Legacy applications would require a program description that clearly states what system components and user files are accessed as a basis for privilege selection. We have looked at

automatically evaluating system call traces from sample application runs. However, often access to system resources depends on dynamic decisions during the program execution. Currently the most promising way to specify such access requirements is through a programmer or experienced user. This issue is not unique to our mechanisms but rather present in any system that aims to achieve least privilege access. In Akenti privilege problems can be debugged by an experienced user via extensive log files. In the case of legacy applications our experiences show that it may be difficult for an arbitrary user to infer which privileges are missing from the often unrelated errors a legacy application reports in the case of insufficient privileges. Legacy applications more often than not were written with an open, relatively unrestricted system in mind.

Our mechanism for privilege management provides better support for accounting of system usage than impersonation schemes as the resource knows from the presented credentials who requests a specific accesses and who authorized the access. When impersonation is used the identity of the requesting entity, which is impersonating the authenticating user, is typically unknown.

A resource gatekeeper authorizing requests using our enforcement mechanism requires super user privileges to assume the user identities of privilege credential issuers and of the authenticated user to perform tasks on their behalf (setting and revoking of privileges, execution of legacy programs). We have not found a way to ease this requirement. However, existing resource gatekeepers that can change their effective user id to the requesting entities local user id (credential mapping) have the same requirement.

In order for our mechanisms to easily scale to larger user groups we are investigating the use of grid information services or standard LDAP servers for the distribution of attribute certificates and information on certificate validity. Furthermore an auditing and logging server, either personal or per community, could be used to supply issuers with unique serial numbers for their attribute certificates. Such a server could then index all issued attribute certificates and thus act as an auditing entity that can supply a list of all privileges that an entity currently holds. This information may be needed when reliable revocation of privileges is important.

File system ACLs and mechanisms to inject DLLs for security evaluation of system calls are also present in current versions of the Microsoft Windows operating system. We plan to investigate these possibilities more closely.

5. Conclusion

This paper reports on our approach to provide enhanced grid security services that support a variety of collaborative scenarios. These services enable low-overhead transient collaboration and improve the abilities of existing systems that focus on more static collaborative structures. More flexible binding of rights is achieved through the use of standard attribute certificates. Low-overhead enforcement mechanisms for file access rights enable efficient realization of fine-grain access policies while supporting legacy software. The ability to securely run legacy codes with fine-grained access permissions using existing operating system extensions is a much-needed feature in grid environments. User-to-user delegation without administrative intervention may prove to be the enabling mechanism for transient ad-hoc collaboration. Our work complements that of GSI and CAS, can interoperate with these systems, and incorporates experiences from the Akenti project.

References

[BAS97] J. Basney, M. Livny, T. Tannenbaum, "High Throughput Computing with Condor", HPCU news, Volume 1(2), June 1997.

[FOS98] I. Foster et al, "A Security Architecture for Computational Grids", ACM Conference Proceedings, Computers and Security, ACM Press, NY, pp. 83-91, 1998

[FOS99] I. Foster, C. Kesselman, "Globus: A Toolkit-Based Grid Architecture" The Grid, Blueprint for a Future Computing Infrastructure, Morgan Kaufmann, San Francisco, 1999, pp. 259-278

[FOS01] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of Supercomputer Applications, 2001.

[FOS02] I. Foster et al, “The Physiology of the Grid – An Open Grid Services Architecture for Distributed Systems Integration”, Draft research paper presented at the Global Grid Forum 4, February 2002, http://www.globus.org/research/papers/ogsa.pdf

[GOL96] I. Goldberg et. al, "A secure environment for untrusted helper applications" Proceedings of the Sixth USENIX UNIX Security Symposium, July 1996

[GRI99] A. Grimshaw et al., “Legion: An Operating System for Wide-Area Computing”, IEEE Computer, 32:5, May 1999: pp. 29-37.

[HAC01] T. Hacker, B. Athey, “A Methodology for Account Management in Grid Computing Environments”, In Proc. Second Int. Workshop on Grid Computing, Denver, USA, November 2001

[LAS01] G. von Laszewski et al., “A Java Commodity Grid Kit”, Concurrency and Computation: Practice and Experience, pages 643-662, Volume 13, Issue 8-9, 2001.

[LOR02] M. Lorch, D. Kafura, “Symphony – A Java-Based Composition and Manipulation Framework for Computational Grids”, accepted for publication in the conference proceedings of ccgrid 2002, Berlin, Germany, May 2002

[MYE01] M Myers et al. „ Online Certificate Status Protocol, version 2 “ IETF PKIX Working Group draft, March 2001, http://www.ietf.org/internet-drafts/draft-ietf-pkix-ocspv2-02.txt

[POS88] „IEEE standard portable operating system interface for computer environments”**,** IEEE Std 1003.1-1988 , 30 Sept. 1988

[PEA02] L. Pearlman et al., “A Community Authorization Service for Group Collaboration”, submitted to the 2002 IEEE Workshop on Policies for Distributed Systems and Networks, http://www.globus.org/Security/CAS/CAS_2002_Submitted.pdf

[RFC1421] J. Linn „Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures “, IETF RFC, February 1993

[RFC3281] S. Farrell, R. Housley, „An Internet Attribute Certificate Profile for Authorization“, IETF RFC, April 2002

[ROM00] M. Romberg "UNICORE: Beyond Web-based Job-Submission" Proceedings of the 42nd Cray User Group Conference, May 22-26,2000, Noordwijk

[RYU99] T.V. Ryutov, G. Gheorghiu and B.C. Neuman “ An Authorization Framework for Metacomputing Applications”, Cluster Computing Journal, Vol. 2 Nr. 2, 1999, pp. 15-175

[SAL75] J. R. Salzer and M. D. Schroeder, "The Protection of Information in Computer Systems", Proceedings of the IEEE, Sept. 1975

[STE97] Stevens, R. et al., "From the I-WAY to the National Technology Grid" Communications of the ACM, vol. 40, pp. 50-61, 1997.

[THA01] D. Thain, M. Livny, "Multiple Bypass: Interposition Agents for Distributed Computing", Journal of Cluster Computing, volume 4, pages 39-47, 2001.

[THO99] M. Thompson et al., “Certificate based Access Control for Widely Distributed Resources”, Proceedings of the 8th Usenix Security Symposium, 1999

[THO01] M. Thompson, “Akenti Policy Language”, White paper, http://www-itg.lbl.gov/Akenti/Papers/, July 2001

[VXE02] The Virtual Execution Environment http://www.intes.odessa.ua/vxe, February 2002

[XACML] Organization for the Advancement of Structured Information Standards (OASIS) “extensible Access Control Markup Language”, http://www.oasis-open.org/committees/xacml/index.shtml, February 21, 2002

[XRML] “XrML - The Digital Rights Language for Trusted Content and Services”, http://www.xrml.org, February 25, 2002